Interview Prep Notes

Prajwal Rao

June 6, 2024

Contents

1 Image processing 101 **2**

1.1 Preparing input . . . 2
1.2 Preparing output . . . 5
1.3 Otsu Thresholding . . . 5
1.4 BGR to HSV . . . 5
1.5 Discretization of Color . . . 6
1.6 Average pooling . . . 6
1.7 Gaussian filter . . . 7
1.8 Median filter . . . 8
1.9 Smoothing filter . . . 8
1.10 Motion Filter . . . 9
1.11 Max Min filter . . . 9
1.12 Differential filter . . . 9
1.13 Sobel Filter . . . 9
1.14 Prewitt Filter . . . 9
1.15 Laplacian Filter . . . 9
1.16 Emboss Filter . . . 9
1.17 Log Filter . . . 9

1.18 Histogram display 9

1.19 Histogram normalization 9

1 Image processing 101

1.1 Preparing input

```
import cv2
import numpy as np
img = cv2.imread("./Image_Processing_100_Questions/Question_01_10/imori.jpg")
print(img)
```

[[file:[[[132 80 67] [104 55 39] [100 54 36] ... [175 109 110] [134 88 70] [163 126 100]]

[[140 88 71] [117 65 52] [106 54 47] ... [177 139 135] [176 137 123] [152 110 91]]

[[137 85 69] [131 77 66] [119 67 60] ... [207 155 148] [232 179 159] [161 104 82]]

...

[[231 172 152] [153 97 80] [160 107 97] ... [107 55 38] [101 60 38] [93 59 36]]

[[255 198 176] [172 114 95] [168 116 104] ... [150 76 58] [119 59 35] [112 58 33]]

[[214 154 130] [187 130 109] [176 124 112] ... [185 117 94] [151 98 71] [116 70 39]]]]]
[[file:[[[132 80 67] [104 55 39] [100 54 36] ... [175 109 110] [134 88 70] [163 126 100]]

[[140 88 71] [117 65 52] [106 54 47] ... [177 139 135] [176 137 123] [152 110 91]]

[[137 85 69] [131 77 66] [119 67 60] ... [207 155 148] [232 179 159] [161 104 82]]

...

[[231 172 152] [153 97 80] [160 107 97] ... [107 55 38] [101 60 38] [93 59 36]]

[[255 198 176] [172 114 95] [168 116 104] ... [150 76 58] [119 59 35] [112 58 33]]

[[214 154 130] [187 130 109] [176 124 112] ... [185 117 94] [151 98 71] [116 70 39]]]]]
[[file:[[[132 80 67] [104 55 39] [100 54 36] ... [175 109 110] [134 88 70] [163 126 100]]

[[140 88 71] [117 65 52] [106 54 47] ... [177 139 135] [176 137 123] [152 110 91]]

[[137 85 69] [131 77 66] [119 67 60] ... [207 155 148] [232 179 159] [161 104 82]]

. . .

[[231 172 152] [153 97 80] [160 107 97] . . . [107 55 38] [101 60 38] [93 59 36]]

[[255 198 176] [172 114 95] [168 116 104] . . . [150 76 58] [119 59 35] [112 58 33]]

[[214 154 130] [187 130 109] [176 124 112] . . . [185 117 94] [151 98 71] [116 70 39]]]]]
[[file:[[[132 80 67] [104 55 39] [100 54 36] . . . [175 109 110] [134 88 70] [163 126 100]]

[[140 88 71] [117 65 52] [106 54 47] . . . [177 139 135] [176 137 123] [152 110 91]]

[[137 85 69] [131 77 66] [119 67 60] . . . [207 155 148] [232 179 159] [161 104 82]]

. . .

[[231 172 152] [153 97 80] [160 107 97] . . . [107 55 38] [101 60 38] [93 59 36]]

[[255 198 176] [172 114 95] [168 116 104] . . . [150 76 58] [119 59 35] [112 58 33]]

[[214 154 130] [187 130 109] [176 124 112] . . . [185 117 94] [151 98 71] [116 70 39]]]]]
[[file:[[[132 80 67] [104 55 39] [100 54 36] . . . [175 109 110] [134 88 70] [163 126 100]]

[[140 88 71] [117 65 52] [106 54 47] . . . [177 139 135] [176 137 123] [152 110 91]]

[[137 85 69] [131 77 66] [119 67 60] . . . [207 155 148] [232 179 159] [161 104 82]]

. . .

[[231 172 152] [153 97 80] [160 107 97] . . . [107 55 38] [101 60 38] [93 59 36]]

[[255 198 176] [172 114 95] [168 116 104] . . . [150 76 58] [119 59 35] [112 58 33]]

[[214 154 130] [187 130 109] [176 124 112] . . . [185 117 94] [151 98 71] [116 70 39]]]]]
[[file:[[[132 80 67] [104 55 39] [100 54 36] . . . [175 109 110] [134 88 70] [163 126 100]]

[[140 88 71] [117 65 52] [106 54 47] . . . [177 139 135] [176 137 123] [152 110 91]]

[[137 85 69] [131 77 66] [119 67 60] . . . [207 155 148] [232 179 159] [161 104 82]]

. . .

[[231 172 152] [153 97 80] [160 107 97] . . . [107 55 38] [101 60 38] [93 59 36]]

[[255 198 176] [172 114 95] [168 116 104] . . . [150 76 58] [119 59 35] [112 58 33]]

[[214 154 130] [187 130 109] [176 124 112] . . . [185 117 94] [151 98 71] [116 70 39]]]]]
[[file:[[[132 80 67] [104 55 39] [100 54 36] . . . [175 109 110] [134 88 70] [163 126 100]]

[[140 88 71] [117 65 52] [106 54 47] ... [177 139 135] [176 137 123] [152 110 91]]

[[137 85 69] [131 77 66] [119 67 60] ... [207 155 148] [232 179 159] [161 104 82]]

...

[[231 172 152] [153 97 80] [160 107 97] ... [107 55 38] [101 60 38] [93 59 36]]

[[255 198 176] [172 114 95] [168 116 104] ... [150 76 58] [119 59 35] [112 58 33]]

[[214 154 130] [187 130 109] [176 124 112] ... [185 117 94] [151 98 71] [116 70 39]]]]]
[[file:[[[132 80 67] [104 55 39] [100 54 36] ... [175 109 110] [134 88 70] [163 126 100]]

[[140 88 71] [117 65 52] [106 54 47] ... [177 139 135] [176 137 123] [152 110 91]]

[[137 85 69] [131 77 66] [119 67 60] ... [207 155 148] [232 179 159] [161 104 82]]

...

[[231 172 152] [153 97 80] [160 107 97] ... [107 55 38] [101 60 38] [93 59 36]]

[[255 198 176] [172 114 95] [168 116 104] ... [150 76 58] [119 59 35] [112 58 33]]

[[214 154 130] [187 130 109] [176 124 112] ... [185 117 94] [151 98 71] [116 70 39]]]]]
[[file:[[[132 80 67] [104 55 39] [100 54 36] ... [175 109 110] [134 88 70] [163 126 100]]

[[140 88 71] [117 65 52] [106 54 47] ... [177 139 135] [176 137 123] [152 110 91]]

[[137 85 69] [131 77 66] [119 67 60] ... [207 155 148] [232 179 159] [161 104 82]]

...

[[231 172 152] [153 97 80] [160 107 97] ... [107 55 38] [101 60 38] [93 59 36]]

[[255 198 176] [172 114 95] [168 116 104] ... [150 76 58] [119 59 35] [112 58 33]]

[[214 154 130] [187 130 109] [176 124 112] ... [185 117 94] [151 98 71] [116 70 39]]]]]
[[file:[[[132 80 67] [104 55 39] [100 54 36] ... [175 109 110] [134 88 70] [163 126 100]]

[[140 88 71] [117 65 52] [106 54 47] ... [177 139 135] [176 137 123] [152 110 91]]

[[137 85 69] [131 77 66] [119 67 60] ... [207 155 148] [232 179 159] [161 104 82]]

...

[[231 172 152] [153 97 80] [160 107 97] ... [107 55 38] [101 60 38] [93 59 36]]

[[255 198 176] [172 114 95] [168 116 104] ... [150 76 58] [119 59 35] [112 58 33]]

[[214 154 130] [187 130 109] [176 124 112] ... [185 117 94] [151 98 71] [116 70 39]]]]]
[[file:[[[132 80 67] [104 55 39] [100 54 36] ... [175 109 110] [134 88 70] [163 126 100]]

[[140 88 71] [117 65 52] [106 54 47] ... [177 139 135] [176 137 123] [152 110 91]]

[[137 85 69] [131 77 66] [119 67 60] ... [207 155 148] [232 179 159] [161 104 82]]

...

[[231 172 152] [153 97 80] [160 107 97] ... [107 55 38] [101 60 38] [93 59 36]]

[[255 198 176] [172 114 95] [168 116 104] ... [150 76 58] [119 59 35] [112 58 33]]

[[214 154 130] [187 130 109] [176 124 112] ... [185 117 94] [151 98 71] [116 70 39]]]]]

1.2 Preparing output

```
def save_output(fname, image):
    cv2.imwrite(fname, image)
    return fname
```

1.3 Otsu Thresholding

1.4 BGR to HSV

Fist the RGB values are normalized by dividing by 255. Therefore, $R\prime = R/255$, $G\prime = G/255$, $B\prime = B/255$. Then we calculate C_{max} and C_{min}. Now $\Delta = C_{max} - C_{min}$. To calculate:

$$C_{max} = max(R\prime, G\prime, B\prime)$$

$$C_{min} = min(R\prime, G\prime, B\prime)$$

C_{max} and C_{min} are the largest and smallest of R, G and B respectively.

For **hue** calculation If $C_{max} = R\prime$, then,

$$H = 60\text{ř} \times (\frac{G\prime - B\prime}{\Delta} mod 6)$$

If $C_{max} = G\prime$, then,

$$H = 60\text{ř} \times (\frac{B\prime - R\prime}{\Delta} + 2)$$

If $C_{max} = B\prime$, then,

$$H = 60\text{ř} \times (\frac{R\prime - G\prime}{\Delta} + 4)$$

For **saturation** calculation, if $C_{max} = 0$,

$$S = 0$$

else if $C_{max} \neq 0$,

$$S = \frac{\Delta}{C_{max}}$$

For value calculation,

$$V = C_{max}$$

1.5 Discretization of Color

Refers to the quantization of color

- Find the quantization levels, here we use 4
- Get the ranges for example for 4 levels: $(-1, 63)$, $(64, 127)$, $(127, 191)$, $(191, 255)$

  ```
  out = img.copy()
  for i in range(4):
      ind = np.where(((64*i-1) <=out) & (out < (64*(i+1)-1)))
      out[ind] = 32* (2*i+1)

  fname = save_output("q6.jpg", out)
  fname
  ```

- Find the midpoint of each quantization levels: 32, 96, 122
- The Value of output is these values where the older values occur in that range

1.6 Average pooling

Image avg pooling keeps its size the same. We can avg/max pool image by doing it in each dim separately in x

- First image chunks Nh and Nw

```
out = img.copy()
H, W, C = img.shape
G = 8
Nh = int(H / G)
Nw = int(W / G)
```

- Then do the pooling

```
for y in range(Nh):
    for x in range(Nw):
        for c in range(C):
            out[G*y:G*(y+1),
                G*x:G*(x+1),
                c] = np.max(out[G*y:G*(y+1),
                             G*x:G*(x+1),
                             c])
fname = save_output("q8.jpg", out)
fname
```

1.7 Gaussian filter

The formula for gaussian filter is

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- First we have to set the filter parameters: size and σ

```
K_size = 3
sigma = 1.3
```

- Do zero padding of the kernel size on each side

  ```
  out = img.copy()
  H, W, C = img.shape
  pad = K_size // 2
  out = np.zeros([H+pad*2, W+pad*2, 3], dtype=np.float32)
  out[pad:pad+H, pad:pad+W] = img.copy().astype(np.float32)
  ```

- Create the filter using the formula **??**.

  ```
  K = np.zeros([K_size, K_size], dtype=np.float32)
  for x in range(-pad, -pad+K_size):
      for y in range(-pad, -pad+K_size):
          K[y+pad, x+pad] = np.exp(-(x**2+y**2)/(2*(sigma**2)))
  K /= (sigma*np.sqrt(2*np.pi))
  K /= K.sum()
  print(K)
  ```

[[file:[[0.08941182 0.12019445 0.08941182] [0.12019445 0.1615749 0.12019445] [0.08941182 0.12019445 0.08941182]]]] [[file:[[0.08941182 0.12019445 0.08941182] [0.12019445 0.1615749 0.12019445] [0.08941182 0.12019445 0.08941182]]]]

- Run this filter on the image and save results

```
for y in range(H):
    for x in range(W):
        for c in range(C):
            out[pad+y, pad+x, c] = np.sum(K* out[y:y+K_size, x:x+K_size, c])
out = out[pad:pad+H, pad:pad+W].astype(np.uint8)
fname = save_output("q9.jpg", out)
fname
```

1.8 Median filter

- Do zero padding of the kernel size on each side

```
K_size = 3
out = img.copy()
H, W, C = img.shape
pad = K_size // 2
out = np.zeros([H+pad*2, W+pad*2, C], dtype=np.float32)
out[pad:pad+H, pad:pad+W] = img.copy().astype(np.float32)
```

- Run the filter over image using `np.median`

```
for y in range(H):
    for x in range(W):
        for c in range(C):
            out[pad+y, pad+x, c] = np.median(out[y:y+K_size, x:x+K_size, c])
out = out[pad:pad+H, pad:pad+W].astype(np.uint8)
fname = save_output("q10.jpg", out)
fname
```

1.9 Smoothing filter

- Do zero padding of kernel size on each side

```
K_size = 3
out = img.copy()
H, W, C = img.shape
```

```
pad = K_size // 2
out = np.zeros([H+pad*2, W+pad*2, C], dtype=np.float32)
out[pad:pad+H, pad:pad+W] = img.copy().astype(np.float32)
tmp = out.copy()

for y in range(H):
    for x in range(W):
        for c in range(C):
            out[pad+y, pad+x, c] = np.mean(tmp[y:y+K_size, x:x+K_size, c])
out = out[pad:pad+H, pad:pad+W].astype(np.uint8)
fname = save_output("q11.jpg", out)
fname
```

1.10 Motion Filter

1.11 Max Min filter

1.12 Differential filter

1.13 Sobel Filter

1.14 Prewitt Filter

1.15 Laplacian Filter

1.16 Emboss Filter

1.17 Log Filter

1.18 Histogram display

- Histogram graph depicts the frequency of pixels.

```
import matplotlib.pyplot as plt
img_dark = cv2.imread("./Image_Processing_100_Questions/Question_11_20/imori_dark.j
plt.hist(img_dark.ravel(), bins=255, rwidth=0.8, range=(0, 255))
plt.savefig("q20.jpg")
"q20.jpg"
```

1.19 Histogram normalization

- Get the histogram of the greyscale image
- calcu